Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**Отчет по лабораторной работе “Классы” №6**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

Выполнил студент гр. ИВТ-21-1б

Ахунов Руслан Булатович

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

г. Пермь - 2022

**Постановка задачи:**

1.  Определить класс-контейнер.

2.  Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.

3.  Перегрузить операции, указанные в варианте.

4.  Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.

5.  Написать тестирующую программу, иллюстрирующую выполнение операций.

Вариант 2:

Базовый класс:

Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции:

[]– доступа по индексу;

int() – определение размера вектора;

+ вектор – сложение элементов векторов a[i]+b[i];

+n - переход вправо к элементу c номером n ( с помощью класса-итератора).

**Анализ задачи:**

class vector

{

private:

    int size;

    int\* cur;

    Iterator beg;

    Iterator end;

public:

    vector(int s, int k = 0)

    {

        size = s;

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = k;

        }

        beg.elem = &cur[0];

        end.elem = &cur[size];

    }

    vector(const vector& tmp)

    {

        size = tmp.size;

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = tmp.cur[i];

        }

        beg = tmp.beg;

        end = tmp.end;

    }

    vector& operator=(const vector& tmp)

    {

        if (this == &tmp)

        {

            return \*this;

        }

        size = tmp.size;

        if (cur != 0)

        {

            delete[]cur;

        }

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = tmp.cur[i];

        }

        beg = tmp.beg;

        end = tmp.end;

        return \*this;

    }

    vector& operator+(const int t)

    {

        vector tmp(size);

        for (int i = 0; i < size; i++)

        {

            tmp.cur[i] += cur[i] + t;

        }

        return tmp;

    }

    int& operator[](int i)

    {

        if (i < size)

        {

            return cur[i];

        }

        else

        {

            cout << "i > размер вектора" << endl;

        }

    }

    int operator()()

    {

        return size;

    }

    friend ostream& operator<<(ostream& out, const vector& tmp);

    friend istream& operator>>(istream& in, vector& tmp);

    Iterator first()

    {

        return beg;

    }

    Iterator last()

    {

        return end;

    }

};

ostream& operator<<(ostream& out, const vector& tmp)

{

    for (int i = 0; i < tmp.size; i++)

    {

        out << tmp.cur[i] << " ";

    }

    return out;

}

istream& operator>>(istream& in, vector& tmp)

{

    for (int i = 0; i < tmp.size; i++)

    {

        in >> tmp.cur[i];

    }

    return in;

}

Класс-контейнер vector, в котором реализовано конструкторы, деструктор, операции ввода-вывода, операцию присваивания. А также были реализованы: []– доступа по индексу;

int() – определение размера вектора;

+ вектор – сложение элементов векторов a[i]+b[i];

+n - переход вправо к элементу c номером n ( с помощью класса-итератора).

Еще были добавлены атрибуты и методы для работы с итераторм:

first – возвращает указатель на первый элемент.

last – возвращает указатель на последний элемент.

Iterator beg – указатель на первый элемент вектора.

Iterator end – указатель на элемент, следующий за последним.

class Iterator

{

private:

    friend class vector;

    int\* elem;

public:

    Iterator()

    {

        elem = 0;

    }

    Iterator(const Iterator& tmp)

    {

        elem = tmp.elem;

    }

    bool operator==(const Iterator& tmp)

    {

        return elem == tmp.elem;

    }

    bool operator!=(const Iterator& tmp)

    {

        return elem != tmp.elem;

    }

    void operator++()

    {

        ++elem;

    }

    void operator--()

    {

        --elem;

    }

    int& operator\*()

    {

        return \*elem;

    }

};

Класс Iterator, в нем есть поле, которое является указателем на элемент типа int. Так же в спецификаторе доступа реализованы конструкторы, так же перегружены операторы сравнения и инкремента, декремента, еще и операция разыменовывания.

int main()

{

    setlocale(LC\_ALL, "Russian");

    vector a(5);

    cout << a << "\n";

    cin >> a;

    cout << a << "\n";

    a[2] = 100;

    cout << a << "\n";

    vector b(10);

    cout << b << "\n";

    b = a;

    cout << b << "\n";

    vector c(10);

    c = b + 100;

    cout << c << "\n";

    cout << "Длина а = " << a() << endl;

    cout << \*(a.first()) << endl;

    Iterator i = a.first();

    ++i;

    cout << \*i << endl;

    for (i = a.first(); i != a.last(); ++i)

    {

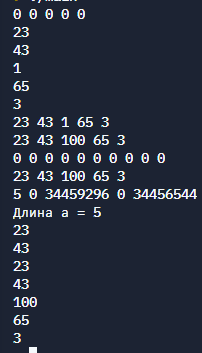
        cout << \*i << endl;

    }

}

В функции main, мы инициализируем класс-контейнер вектор переменной а. Проверяем операции [], =, +. Переменную типа Iterator устанавливаем на первый элемент вектора а, с помощью метода first. Выполняем операцию инкремента, разыменовывания и выводим значения элементов вектора с помощью итератора.

**Результат работы программы**

****

Выводиться пустой вектор размера 5. Заполняем его значениями и выводим снова, создаем другие вектора и проводим различные операции.

**Код программы**

#include <iostream>

using namespace std;

class Iterator

{

private:

    friend class vector;

    int\* elem;

public:

    Iterator()

    {

        elem = 0;

    }

    Iterator(const Iterator& tmp)

    {

        elem = tmp.elem;

    }

    bool operator==(const Iterator& tmp)

    {

        return elem == tmp.elem;

    }

    bool operator!=(const Iterator& tmp)

    {

        return elem != tmp.elem;

    }

    void operator++()

    {

        ++elem;

    }

    void operator--()

    {

        --elem;

    }

    int& operator\*()

    {

        return \*elem;

    }

};

class vector

{

private:

    int size;

    int\* cur;

    Iterator beg;

    Iterator end;

public:

    vector(int s, int k = 0)

    {

        size = s;

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = k;

        }

        beg.elem = &cur[0];

        end.elem = &cur[size];

    }

    vector(const vector& tmp)

    {

        size = tmp.size;

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = tmp.cur[i];

        }

        beg = tmp.beg;

        end = tmp.end;

    }

    vector& operator=(const vector& tmp)

    {

        if (this == &tmp)

        {

            return \*this;

        }

        size = tmp.size;

        if (cur != 0)

        {

            delete[]cur;

        }

        cur = new int[size];

        for (int i = 0; i < size; i++)

        {

            cur[i] = tmp.cur[i];

        }

        beg = tmp.beg;

        end = tmp.end;

        return \*this;

    }

    vector& operator+(const int t)

    {

        vector tmp(size);

        for (int i = 0; i < size; i++)

        {

            tmp.cur[i] += cur[i] + t;

        }

        return tmp;

    }

    int& operator[](int i)

    {

        if (i < size)

        {

            return cur[i];

        }

        else

        {

            cout << "i > размер вектора" << endl;

        }

    }

    int operator()()

    {

        return size;

    }

    friend ostream& operator<<(ostream& out, const vector& tmp);

    friend istream& operator>>(istream& in, vector& tmp);

    Iterator first()

    {

        return beg;

    }

    Iterator last()

    {

        return end;

    }

};

ostream& operator<<(ostream& out, const vector& tmp)

{

    for (int i = 0; i < tmp.size; i++)

    {

        out << tmp.cur[i] << " ";

    }

    return out;

}

istream& operator>>(istream& in, vector& tmp)

{

    for (int i = 0; i < tmp.size; i++)

    {

        in >> tmp.cur[i];

    }

    return in;

}

int main()

{

    setlocale(LC\_ALL, "Russian");

    vector a(5);

    cout << a << "\n";

    cin >> a;

    cout << a << "\n";

    a[2] = 100;

    cout << a << "\n";

    vector b(10);

    cout << b << "\n";

    b = a;

    cout << b << "\n";

    vector c(10);

    c = b + 100;

    cout << c << "\n";

    cout << "Длина а = " << a() << endl;

    cout << \*(a.first()) << endl;

    Iterator i = a.first();

    ++i;

    cout << \*i << endl;

    for (i = a.first(); i != a.last(); ++i)

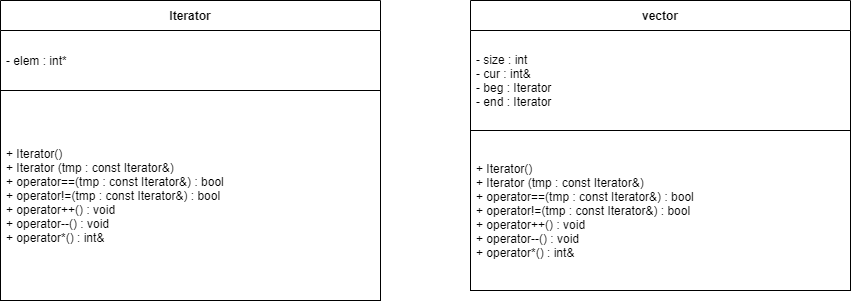
    {

        cout << \*i << endl;

    }

}

**Диаграмма класса**



**Вопросы**

**1. Что такое абстрактный тип данных? Привести примеры АТД.**

АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

Пример: контейнер, класс

**2. Привести примеры абстракции через параметризацию.**

Примером абстракции через параметризацию является концепция «функция», передаваемые параметры которой являются формальными, а фактические связываются с ними в момент использования такой абстракции.

**3. Привести примеры абстракции через спецификацию.**

Мы пользуемся абстракцией через спецификацию всякий раз, когда связываем с процедурой некий комментарий, достаточно информативный для того, чтобы иметь возможность работать без анализа тела процедуры. Спецификация описывает соглашение между разработчиками и пользователями. Разработчик берется написать модуль, а пользователь соглашается не полагаться на знания о том, как именно этот модуль реализован, т.е. не предполагать ничего такого, что не было бы указано в спецификации

**4. Что такое контейнер? Привести примеры.**

Контейнер — это набор некоторого количества обязательно однотипных элементов, упакованных в контейнер определённым образом. Простейшим прототипом контейнера в классическом языке C++ является массив.

**5. Какие группы операций выделяют в контейнерах?**

* + - Операции доступа к элементам, которые обеспечивают и операцию замены значений элементов;
    - Операции добавления и удаления элементов или групп элементов;
    - Операции поиска элементов и групп элементов;
    - Операции объединения контейнеров;
    - Специальные операции, которые зависят от вида контейнера.

**6. Какие виды доступа к элементам контейнера существуют? Привести примеры.**

Доступ к элементам контейнера бывает: последовательный, прямой и ассоциативный.

- Прямой доступ – это доступ по индексу. Например, a[10] – требуется найти элемент контейнера с номером 10.

- Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Индексом может служить слово, например, a[“word”]

- При последовательном доступе осуществляется перемещение от элемента к элементу контейнера. Набор операций последовательного доступа включает следующие:

* + - * Перейти к первому элементу;
      * Перейти к последнему элементу;
      * Перейти к следующему элементу;
      * Перейти к предыдущему элементу;
      * Перейти на n элементов вперед;
      * Перейти на n элементов назад;
      * Получить текущий элемент.

**7. Что такое итератор?**

Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера.

**8. Каким образом может быть реализован итератор?**

Итератор может быть реализован как часть класса-контейнера в виде набора методов: v.first, v.last, v.next, v.prev, v.skip и т.д.

**9. Каким образом можно организовать объединение контейнеров?**

• Простое сцепление двух контейнеров: в новый контейнер попадают сначала элементы первого контейнера, потом второго, операция не коммутативна.

• Объединение упорядоченных контейнеров, новый контейнер тоже будет упорядочен, операция коммутативна.

• Объединение контейнеров как объединение множеств, в новый контейнер попадают только те элементы, которые есть хотя бы в одном контейнере, операция коммутативна.

• Объединение контейнеров как пересечение множеств, в новый контейнер попадают только те элементы, которые есть в обоих контейнерах, операция коммутативна.

• Для контейнеров-множеств может быть еще реализована операция вычитания, в контейнер попадают только те элементы первого контейнера, которых нет во втором, операция не коммутативна.

• Извлечение части элементов из контейнера и создание нового контейнера. Эта операция может быть выполнена с помощью конструктора, а часть контейнера задается двумя итераторами.

**10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?**

Ассоциативный доступ.

**11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?**

Стек

**12. Какой из объектов (a,b,c,d) является контейнером?**

a. int mas=10;

b. int mas;

c. struct {char name[30]; int age;} mas;

d. int mas[100]; /////////////////////

**13. Какой из объектов (a,b,c,d) не является контейнером?**

a. int a[]={1,2,3,4,5};

b. int mas[30];

c. struct {char name[30]; int age;} mas[30]; ////////////////

d. int mas;

**14. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?**

Прямой доступ

**15. Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?**

Последовательный